

# An Experimental Evaluation of the Best-of-Many Christofides' Algorithm for the Traveling Salesman Problem\*

Kyle Genova<sup>†</sup>      David P. Williamson<sup>‡</sup>

June 26, 2015

## Abstract

Recent papers on approximation algorithms for the traveling salesman problem (TSP) have given a new variant on the well-known Christofides' algorithm for the TSP, called the *Best-of-Many Christofides' algorithm*. The algorithm involves sampling a spanning tree from the solution the standard LP relaxation of the TSP, subject to the condition that each edge is sampled with probability at most its value in the LP relaxation. One then runs Christofides' algorithm on the tree by computing a minimum-cost matching on the odd-degree vertices in the tree, and shortcutting the resulting Eulerian graph to a tour. In this paper we perform an experimental evaluation of the Best-of-Many Christofides' algorithm to see if there are empirical reasons to believe its performance is better than that of Christofides' algorithm. Furthermore, several different sampling schemes have been proposed; we implement several different schemes to determine which ones might be the most promising for obtaining improved performance guarantees over that of Christofides' algorithm. In our experiments, all of the implemented methods perform significantly better than the Christofides' algorithm; an algorithm that samples from a maximum entropy distribution over spanning trees seems to be particularly good, though there are others that perform almost as well.

## 1 Introduction

In the traveling salesman problem (TSP), we are given a complete, undirected graph  $G = (V, E)$  as input with costs  $c_e \geq 0$  for all  $e \in E$ , and we must find a tour through all the vertices of minimum cost. In what follows, we will assume that the costs obey the triangle inequality; that is,  $c_{(u,w)} \leq c_{(u,v)} + c_{(v,w)}$  for all  $u, v, w \in V$ . We will sometimes refer to the *asymmetric* traveling salesman problem (ATSP), in which the input is a complete directed graph, and possibly  $c_{(u,v)} \neq c_{(v,u)}$ .

In 1976, Christofides [7] gave a  $\frac{3}{2}$ -approximation algorithm for the TSP; an  $\alpha$ -approximation algorithm for the TSP is one that runs in polynomial time and returns a solution of cost at most  $\alpha$  times the cost of an optimal solution. The value  $\alpha$  is sometimes known as the performance

---

\*An extended abstract of this paper is scheduled to appear in the 2015 European Symposium on Algorithms (ESA 2015).

<sup>†</sup>Address: Department of Computer Science, Cornell University, Ithaca, NY 14853. Email: [kag278@cornell.edu](mailto:kag278@cornell.edu). Supported in part by NSF grant CCF-1115256.

<sup>‡</sup>Address: School of Operations Research and Information Engineering, Cornell University, Ithaca, NY 14853. Email: [dpr@cs.cornell.edu](mailto:dpr@cs.cornell.edu). Supported in part by NSF grant CCF-1115256.

guarantee of the algorithm. Christofides' algorithm works as follows: it computes a minimum-cost spanning tree (MST)  $F$  of the input graph  $G$ , then finds a minimum-cost perfect matching  $M$  on all the odd-degree vertices of the tree  $F$ . The resulting edge set  $F \cup M$  is then an Eulerian subgraph of  $G$ : it is connected (because  $F$  is connected) and has even degree at all vertices (because we added one edge incident to all odd-degree vertices of  $F$ ). It is well-known that an Eulerian subgraph has an Eulerian traversal that starts at any vertex of the graph, visits all of the edges of the Eulerian subgraph, then returns to the starting vertex. By "shortcutting" any previously visited vertex during the traversal, we can obtain a tour that visits each vertex exactly once and has cost no greater than the cost of the edges in  $F \cup M$ .

No approximation algorithm with performance guarantee better than  $\frac{3}{2}$  is yet known for the TSP. However, some progress has been made in recent years for special cases and variants of the problem. Asadpour, Goemans, Madry, Oveis Gharan, and Saberi [5] gave an  $O(\log n / \log \log n)$ -approximation algorithm for the ATSP (where  $n = |V|$ ), improving on a long-standing  $O(\log n)$ -approximation algorithm of Frieze, Galbati, and Maffioli [10]. A sequence of improvements has been obtained in the special case of the *graph* TSP, in which the input to the problem is an undirected, not necessarily complete graph  $G$ , and the cost  $c_{(u,v)}$  for each  $u, v \in V$  is the number of edges in the shortest  $u$ - $v$  path in  $G$ . In this case, Oveis Gharan, Saberi, and Singh [25] were able to improve slightly on the factor of  $\frac{3}{2}$ . Mömke and Svensson [19] then gave a 1.462-approximation algorithm; Mucha [20] improved the analysis of the Mömke and Svensson algorithm to obtain a  $\frac{13}{9}$ -approximation algorithm. Sebő and Vygen [31], by adding some additional ideas, gave a 1.4-approximation algorithm for graph TSP.

Another line of work considers the *s-t path TSP*. In this problem, we have the same input as the TSP, plus additional vertices  $s, t \in V$ . The goal is to find a minimum-cost Hamiltonian path starting at  $s$  and ending at  $t$  (that is, a path that starts at  $s$ , ends at  $t$ , and visits all other vertices in between). In 1991, Hoogeveen [13] showed that the natural analog of Christofides' algorithm for the *s-t* TSP path problem is a  $\frac{5}{3}$ -approximation algorithm; this remained the best known performance guarantee for over 20 years. In 2012, An, Kleinberg, and Shmoys [3] gave a  $\frac{1+\sqrt{5}}{2}$ -approximation algorithm for the problem; Sebő [30] improved the analysis of the An et al. algorithm to obtain a 1.6-approximation algorithm. Gao [11] gives a nice unification of these two results. Very recently, Vygen [34] gave a slight improvement to a 1.599-approximation algorithm.

A crucial idea in several of these results is to run Christofides' algorithm, but to start with a tree that is determined by an LP relaxation of the problem at hand, rather than the minimum-cost spanning tree. For the TSP, a well-known relaxation of the problem is as follows.

$$\begin{aligned} & \text{Min} \quad \sum_{e \in E} c_e x_e \\ & \text{subject to:} \\ & \quad x(\delta(v)) = 2, \quad \forall v \in V, \\ & \quad x(\delta(S)) \geq 2, \quad \forall S \subset V, S \neq \emptyset, \\ & \quad 0 \leq x_e \leq 1, \quad \forall e \in E, \end{aligned}$$

where  $\delta(S)$  is the set of all edges with exactly one endpoint in  $S$  and we use the shorthand that  $x(F) = \sum_{e \in F} x_e$ . This LP relaxation is sometimes called the *Subtour LP*. It is not hard to show that given a feasible solution  $x$  to the Subtour LP,  $\frac{n-1}{n}x$  is feasible for the spanning tree polytope  $\{x \in \mathbb{R}^{|E|} : x(E) = n - 1, x(E(S)) \leq |S| - 1 \ \forall S \subseteq V, |S| \geq 2\}$ , where  $E(S)$  is the set of all

edges with both endpoints in  $S$ . Similarly, any feasible solution  $x$  to a related relaxation of the  $s$ - $t$  path TSP is also feasible for the spanning tree polytope. Oveis Gharan, Saberi, and Singh [25] propose an algorithm which has since been called (by [3]) *Best-of-Many Christofides*: given the LP solution  $x^*$ , we can compute in polynomial time a decomposition of  $x^*$  into a convex combination of spanning trees, and we run Christofides’ algorithm (or the Hoogeveen variant for  $s$ - $t$  path TSP) for each of these trees and output the lowest cost solution found. More precisely, if  $\chi_F \in \{0, 1\}^{|E|}$  is the characteristic vector of a set of edges  $F$ , then given Subtour LP solution  $x^*$ , we find spanning trees  $F_1, \dots, F_k$  such that  $\frac{n-1}{n}x^* = \sum_{i=1}^k \lambda_i \chi_{F_i}$  for  $\lambda_i \geq 0$  and  $\sum_{i=1}^k \lambda_i = 1$ . Then for each tree  $F_i$  we find a matching  $M_i$  of the odd-degree vertices, and we compute a tour by shortcutting  $F_i \cup M_i$ . We return the cheapest tour found.

An alternative perspective is to consider randomly sampling a spanning tree from a distribution on spanning trees given by the convex combination, then run Christofides’ algorithm on the resulting tree found; that is, we sample tree  $F_i$  with probability  $\lambda_i$ . This perspective potentially allows us to avoid computing the convex combination explicitly. However, the distribution of trees then depends on the (implicit) convex combination. Asadpour et al. [5] and Oveis Gharan, Saberi, and Singh [25] use a *maximum entropy* distribution. For the Asadpour et al. ATSP result, the main property used of the maximum entropy distribution is that in some cuts of edges, the appearance of arcs is negatively correlated. Chekuri, Vondrák, and Zenklusen [6] show how to draw a sample with the appropriate negative correlation properties given an explicit convex combination of trees; their distribution over trees is not the same as the maximum entropy distribution.

An exciting possible direction for an improved approximation algorithm for the TSP is to show that some variation of the Best-of-Many Christofides’ algorithm gives a performance guarantee strictly better than  $3/2$ , either by starting with a convex combination of spanning trees, or using some of the stronger properties obtained by sampling a tree from a negatively correlated distribution, or the maximum entropy distribution. In this paper, we experimentally evaluate these different versions of the Best-of-Many Christofides’ algorithm in order to see if there is empirical evidence that these algorithmic variants are any better than the standard Christofides’ algorithm, and whether any of the variants is more promising than the others.

In particular, we start by implementing Christofides’ algorithm. Since most of our instances are geometric, we compute a Delaunay triangulation using the package Triangle [32]; it is known that the edges of an MST for a 2D Euclidean instance are a subset of the edges of the Delaunay triangulation. We use Prim’s algorithm to compute the MST from these edges. For non-geometric instances, we use Prim’s algorithm to compute the MST. We then use the Blossom V code of Kolmogorov [15] to find a minimum-cost perfect matching on the odd degree vertices of the tree. We compute a tour by shortcutting the resulting Eulerian graph; we perform a simple optimization on the shortcutting. We then use the Concorde TSP solver [4] to compute a solution  $x$  to the subtour LP. We implement two different ways of finding an explicit convex combination of trees equal to  $\frac{n-1}{n}x$ ; in the first, we use a column generation technique suggested by An in his Ph.D. thesis [2] in conjunction with the linear programming solver Gurobi [22]. In the second, we compute a packing of spanning trees via iteratively “splitting off” edges of the LP solution from vertices, then maintaining a convex combination of trees as we “lift back” the split-off edges. For this algorithm, we use a subroutine of Nagamochi and Ibaraki [21] to obtain a complete splitting-off of a vertex. We also implement two methods for obtaining a randomly sampled tree from the support of the LP solution. We first implement the SwapRound procedure of Chekuri et al. [6]; given an explicit convex combination of trees generated by the first two methods, we can sample a spanning tree such that the edges of the tree appearing in any given set are negatively correlated (we define the

negative correlation more precisely in Section 2.3). We also implement the method for computing a maximum entropy distribution over spanning trees given the LP solution  $x^*$ , and then drawing a sample from this distribution, as given in the ATSP paper of Asadpour et al. [5] and the Ph.D. thesis of Oveis Gharan [23]. Our implementation choices for the maximum entropy routine were influenced by a code shared with us by Oveis Gharan [24].

To test our results, we ran these algorithms on TSPLIB instances of Reinelt [26] (both Euclidean and non-Euclidean instances) and Euclidean VLSI instances from Rohe [27]. We also considered graph TSP instances to see if the performance of the algorithms was better for such instances than for weighted instances. For our graph TSP instances, we used undirected graphs from the Koblenz Network Collection of Kunegis [16].

It is known that the standard Christofides' algorithm typically returns solutions of cost of about 9-10% away from the cost of an optimal solution on average (see, for instance, Johnson and McGeoch [14]); this is better than its worst-case guarantee of at most 50% away from the cost of an optimal solution, but not as good as other heuristics (such as the Lin-Kernighan heuristic [17]) that do not have performance guarantees. We confirm these results for the standard Christofides' algorithm on geometric instances, but Christofides' algorithm appears to do worse on graph TSP instances; we had solutions of about 12% away from optimal. All of the Best-of-Many Christofides' algorithms performed substantially better than the standard Christofides' algorithm, with solutions of cost about 3-7% away from optimal for the Euclidean instances, 2-3% away from optimal for the non-Euclidean instances, and under 1% away from optimal for the graph TSP instances. These results may indicate that the graph TSP instances are easier for LP-based algorithms than geometric instances. The algorithm that used the maximum entropy distribution on average outperformed the other Best-of-Many Christofides' algorithms; however, the algorithm that found a convex combination of spanning trees via splitting off, then used the SwapRound routine, was nearly as good as maximum entropy sampling, and was better in some cases.

A very recent example of Schalekamp and van Zuylen [29] suggests that running the Best-of-Many Christofides' algorithm on a fixed convex combination of trees cannot be better than a  $\frac{3}{2}$ -approximation algorithm; this may help explain why the random sampling methods perform better in practice. We discuss this matter further in our concluding section.

Our paper is structured as follows. In Section 2, we give a more detailed description of the algorithms that we implemented. In Section 3, we describe the TSP datasets we used and our machine environment, and in Section 4 we give the results of our experiments, as well as some analysis. We conclude in Section 5.

## 2 Algorithms

In this section, we give descriptions of the various algorithms that we implemented. Sections 2.1 and 2.2 describe the two algorithms that generate explicit convex combinations of spanning trees given the Subtour LP solution  $x^*$ ; as described above, we compute the Subtour LP solution by using the Concorde TSP solver [4]. Section 2.3 describes the SwapRound algorithm of Chekuri, Vondrák, and Zenklusen [6] that generates a randomly sampled tree given an explicit convex combination of spanning trees. Section 2.4 describes the maximum entropy distribution on spanning trees, the algorithm used to compute it, and the algorithm used to draw a sample from this distribution.

As described in the introduction, we also have an implementation of the basic Christofides' algorithm, in which we start with the minimum-cost spanning tree given by Prim's algorithm;

for the two-dimensional geometric instances, we first calculate the Delaunay triangulation using Triangle [32] and run Prim's algorithm on these edges. For the graph TSP instances, because the graphs are connected, the trees will always have a cost of  $n - 1$ . We added a very small random weight to each edge for these instances so that the algorithm chooses a random spanning tree of the graph. We then find a minimum-cost perfect matching by using the Blossom V code of Kolmogorov [15]. In our shortcutting, we build up a tour; if we encounter a vertex  $v$  in the Eulerian traversal that is already part of the tour, we choose whether inserting  $v$  at this point in the tour and removing it from the previous point would be cheaper than skipping over  $v$  at this point in the tour and going on to the next vertex in the traversal.

## 2.1 Column Generation

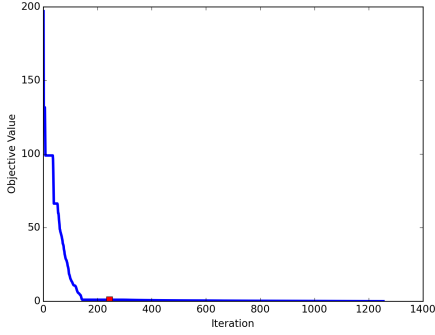
Our first algorithm for decomposing the Subtour LP solution  $\frac{n-1}{n}x^*$  into a convex combination of spanning trees follows an algorithm described by An in his Ph.D. thesis [2]; we use column generation to generate the trees in the convex combination. In particular, we would like to solve the following linear program, in which we have a variable  $y_T$  for each possible spanning tree  $T$ , where we assume that  $x_e^*$  is the given solution to the Subtour LP and the graph  $G = (V, E)$  is on the edges  $E = \{e : x_e^* > 0\}$ :

$$\begin{aligned} & \text{Min} \quad \sum_{e \in E} s_e \\ & \text{subject to:} \\ & \quad \sum_{T: e \in T} y_T + s_e = \frac{n-1}{n} x_e^*, \quad \forall e \in E, \\ & \quad y_T \geq 0, \quad \forall T. \\ & \quad s_e \geq 0, \quad \forall e \in E. \end{aligned}$$

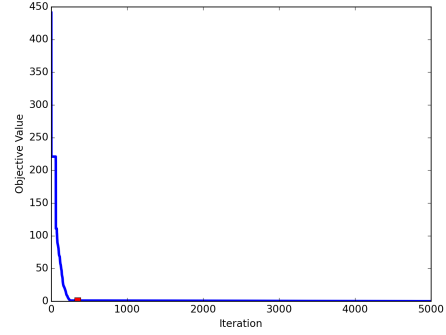
Since  $\frac{n-1}{n}x^*$  can be expressed as a convex combination of spanning trees, the optimal solution to the LP is zero, and for an optimal solution  $(y^*, s^*)$ ,  $y^*$  gives a convex combination of spanning trees. The dual of this LP is:

$$\begin{aligned} & \text{Max} \quad \frac{n-1}{n} \sum_{e \in E} x_e^* z_e \\ & \text{subject to:} \\ & \quad \sum_{e \in T} z_e \leq 0, \quad \forall T, \\ & \quad z_e \leq 1, \quad \forall e \in E. \end{aligned}$$

Because there are too many variables  $y_T$  to include initially, we use a column generation approach. We initially include some variables  $y_T$  representing some spanning trees in the support of the  $x_e^*$ , solve the LP, and find the dual variables  $z_e$ . We then compute a maximum-cost spanning tree  $\hat{T}$  using the value  $z_e$  as the cost of the edge  $e$  for each edge  $e \in E$ . If this cost  $\sum_{e \in \hat{T}} z_e \leq 0$ , then the current solution is optimal; otherwise, we add a new variable  $y_{\hat{T}}$  corresponding to the tree  $\hat{T}$ . We use Gurobi [22] to solve the LPs and Prim's algorithm to compute the maximum-cost spanning trees.



(a) TSPLIB D198.



(b) TSPLIB PCB442.

Figure 1: Column generation objective function value versus iteration count for some sample data sets, run both with and without our cutoff procedure. The small rectangle indicates the iteration in which our cutoff procedure terminates the column generation algorithm early.

As is typical of column generation algorithms, solving the LP to optimality can take a long time. On instances of 500 cities, it could take up to 10 hours of computation time. Thus in order to make comparisons with other methods, we terminated early. In particular, we store the current objective function value and wait until either the value drops by .1 or 100 iterations have occurred. If the objective has not dropped by .1 in the 100 iterations, we terminate; otherwise, once it has dropped by .1 we restart the iteration count. Our cutoff behavior allows us to avoid the long set of iterations in which the method is near optimal but only makes incremental progress; see Figure 1 for an illustration of how the cutoff helps for two instances.

## 2.2 Splitting Off and Tree Packing

Let  $x^*$  be a basic feasible solution to the Subtour LP. It is known that  $x^*$  is rational, so that there exists a  $K$  such that  $Kx^*$  is integer. We let  $\hat{G}$  be a multigraph with  $Kx_e^*$  copies of edge  $e$ . Then by the constraints of the Subtour LP, each vertex has  $2K$  edges incident on it, and for each  $i, j \in V$ , there are at least  $2K$  edge-disjoint paths between  $i$  and  $j$ . Lovász [18] showed that given any edge  $(x, z)$  incident on  $z$  it is possible to find another edge  $(y, z)$  incident on  $z$  such that if we remove  $(x, z)$  and  $(y, z)$  from  $\hat{G}$  and add edge  $(x, y)$ , then for all pairs of vertices  $i, j \in V$ ,  $i, j \neq z$ , there are still  $2K$  edge-disjoint paths between  $i$  and  $j$ . Removing  $(x, z)$  and  $(y, z)$  and adding  $(x, y)$  so as to preserve connectivity in this way is called the *splitting off* operation. A *complete splitting off* at  $z$  removes all  $2K$  edges incident at  $z$  from  $\hat{G}$  and adds  $K$  edges not incident on  $z$  to  $\hat{G}$  so that for all pairs of vertices  $i, j \in V$ ,  $i, j \neq z$ , there are still  $2K$  edge-disjoint paths between  $i$  and  $j$ . Nagamochi and Ibaraki [21] give an  $O(nm \log n + n^2 \log^2 n)$  time algorithm for obtaining a complete splitting off at a node.

We implemented the Nagamochi-Ibaraki algorithm, and use it to obtain a set of  $K$  trees in the original multigraph  $\hat{G}$  as follows (following Frank [9, Chapter 10]). We first iterate through all but two nodes of the graph, obtain a complete splitting off at each node, and keep track of the splitting off in each case (that is, we remember that edge  $(x, y)$  results from splitting off  $(x, z)$  and  $(y, z)$ ). We then inductively construct a set of  $K$  trees, starting with the two nodes, and adding back a

node at a time in reverse of the order in which we did the splitting off operations. The base case is simple: for the two nodes, there will be  $2K$  edges remaining that join them. We choose  $K$  edges arbitrarily, so that we now have  $K$  spanning trees on the two nodes. The remaining edges we call *leftover edges*. In the inductive step, we have  $K$  trees spanning a set of nodes and  $K$  leftover edges; we wish to extend the trees to also span a new node  $z$  by undoing the splitting off operations from node  $z$ . There are  $K$  edges that were split off from  $z$ ; these  $K$  edges appear in the current set of trees or in the set of leftover edges. We maintain a set of *leaf edges*  $F$  in the step. We start by considering any split off edge  $(x, y)$  that is a leftover edge; we replace  $(x, y)$  with  $(x, z)$  in the set of leftover edges, and add  $(y, z)$  to the set of leaf edges. For any tree that contains at least one split off edge  $(x, y)$ , we remove  $(x, y)$  from the tree and add  $(x, z)$  and  $(z, y)$  to the tree. If the same tree contains other split off edges  $(x', y')$ , then for each such edge we calculate the shortest path distance in the tree from  $z$  to all other nodes in the tree. We remove  $(x', y')$  and add either  $(x', z)$  or  $(y', z)$ , depending on whether  $x'$  or  $y'$  is farther away from  $z$  (note that because it is a tree, they cannot be at the same distance); adding the edge with the farther endpoint from  $z$  maintains the connectivity of the tree. The other edge that is not added to the tree ( $(x', z)$  or  $(y', z)$ ) we add to the set of leaf edges. Finally, for each tree that contains no split off edges, we remove an arbitrary edge from the set of leaf edges and add it to the tree. At the end of the process for node  $z$ , we have  $K$  trees that also span  $z$ , have no remaining leaf edges, and have exactly  $K$  leftover edges. Once we have iterated through all the nodes, we will have a set of  $K$  trees and  $K$  leftover edges such that if  $\chi_{F_i}$  is the characteristic vector of the  $i$ th tree and  $\chi_L$  is the characteristic vector of the multiset of leftover edges, then  $\frac{1}{K} \sum_{i=1}^K \chi_{F_i} + \frac{1}{K} \chi_L = x^*$ . Thus  $\frac{1}{K} \sum_{i=1}^K \chi_{F_i} \leq x^*$ .

### 2.3 SwapRound and Negatively Correlated Distributions

The algorithms of the previous two sections give a convex combination of spanning trees such that the convex combination of the characteristic vectors of the trees is dominated by the Subtour LP solution  $x^*$ . Let  $z^* = \sum_{i=1}^K \lambda_i \chi_{F_i} \leq x^*$  give the convex combination of the characteristic vectors  $\chi_{F_i}$  of  $K$  trees  $F_i$ . One can think about the convex combination as being a distribution on spanning trees: we sample tree  $F_i$  with probability  $\lambda_i$ . A nice feature of this sampling scheme is that the expected cost of the sampled tree is  $\sum_{e \in E} c_e z_e^* \leq \sum_{e \in E} c_e x_e^*$ , at most the value of the Subtour LP. This follows since the probability that a given edge  $e \in E$  is in the sampled tree  $F$  is

$$\Pr[e \in F] = \sum_{i: e \in F_i} \lambda_i = \sum_{i: e \in F_i} \lambda_i \chi_{F_i}(e) = z_e^* \leq x_e^*.$$

Let  $X_e$  be a random variable which is 1 if edge  $e$  is in the sampled tree and 0 otherwise; then we have shown that  $E[X_e] = z_e^*$ .

Asadpour et al. [5] show that for proving results about the asymmetric TSP, it is useful to think about drawing a sample such that the edges of the spanning tree appearing in a fixed set are *negatively correlated*. We will say that a probability distribution is negatively correlated if  $E[X_e] = z_e^*$  and for any set of edges  $A \subseteq E$ ,  $E[\prod_{e \in A} X_e] \leq \prod_{e \in A} z_e^*$ , and  $E[\prod_{e \in A} (1 - X_e)] \leq \prod_{e \in A} (1 - z_e^*)$ . Negative correlation allows the proof of concentration bounds that get used in the result of Asadpour et al.

Chekuri, Vondrák, and Zenklusen [6] give a sampling scheme they call SwapRound such that given any convex combination of spanning trees as input, SwapRound gives a sample from a negatively correlated distribution as output (their result applies more generally to matroids). Let  $F_1, \dots, F_k$  be the trees from the convex combination. The algorithm maintains a spanning tree

$F$ , which is initially  $F_1$ . Then it loops through the other trees  $F_2, \dots, F_k$ , and calls a subroutine, MergeBasis, with the two trees  $F$  and  $F_i$ , and with probability weights  $\sum_{j=1}^{i-1} \lambda_j$  and  $\lambda_i$ , and updates  $F$  to be the result of MergeBasis. The routine MergeBasis, given two trees  $F$  and  $F'$  and weights  $\lambda$  and  $\lambda'$ , repeatedly interchanges edges between the trees  $F$  and  $F'$  until the two are the same. While  $F \neq F'$ , the routine finds edges  $e \in F - F'$  and  $e' \in F' - F$  such that  $F - e + e'$  is a spanning tree and  $F' - e' + e$  is a spanning tree (such edges are known to exist if  $F \neq F'$ ). Then with probability  $\lambda/(\lambda + \lambda')$ , the routine updates  $F'$  to  $F' - e' + e$ , and otherwise the routine sets  $F$  to  $F - e + e'$ . When  $F = F'$ , the routine returns  $F$ .

We implemented the SwapRound and MergeBasis routines in order to see if sampling a spanning tree from a negatively correlated distribution would lead to better overall results. Because both the column generation and splitting off methods give a convex combination of spanning trees, we tried these two methods both with and without the SwapRound routine on the output. Because the sampling can be performed in parallel, we had four threads running to draw the samples. We drew 1000 samples per instance, and output the best tour found.

## 2.4 The Maximum Entropy Distribution

Asadpour et al. [5] consider sampling spanning trees from the maximum entropy distribution over spanning trees. Given the subtour LP solution  $x^*$ , we set  $z^* = \frac{n-1}{n}x^*$ . If  $\mathcal{T}$  is the set of all spanning trees of the graph  $G$ , then the maximum entropy distribution is an optimal solution to the following:

$$\begin{aligned} & \text{Inf} \quad \sum_{T \in \mathcal{T}} p(T) \log p(T) \\ & \text{subject to:} \\ & \quad \sum_{T: e \in T} p(T) = z_e^*, \quad \forall e \in E, \\ & \quad \sum_{T \in \mathcal{T}} p(T) = 1 \\ & \quad p(T) \geq 0, \quad \forall T. \end{aligned}$$

Asadpour et al. show that the constraint  $\sum_{T \in \mathcal{T}} p(T) = 1$  is redundant. Given that  $z^*$  is in the relative interior of the spanning tree polytope, they argue that there must exist  $\gamma_e^*$  for all  $e \in E$  such that sampling tree  $T$  with probability proportional to  $p(T) = e^{\gamma^*(T)}$  (with  $\gamma^*(T) \equiv \sum_{e \in T} \gamma_e^*$ ) results in  $\Pr[e \in T] = z_e^*$  and gives the maximum entropy distribution.

Asadpour et al. then give an algorithm for computing values  $\tilde{\gamma}_e$  that approximately satisfy the conditions. In particular, the value  $\tilde{\gamma}_e$  are such that if we set

$$\tilde{p}(T) \equiv \frac{1}{P} \exp \left( \sum_{e \in T} \tilde{\gamma}_e \right), \text{ where } P \equiv \sum_{T \in \mathcal{T}} \exp \left( \sum_{e \in T} \tilde{\gamma}_e \right),$$

then

$$\tilde{z}_e \equiv \sum_{T \in \mathcal{T}: T \ni e} \tilde{p}(T) \leq (1 + \epsilon) z_e^*.$$

To compute the  $\tilde{\gamma}_e$ , we use a combination of the algorithm suggested in Asadpour et al. and one given by code written by Oveis Gharan [24]. We use the Eigen C++ template library [8] to perform matrix operations needed by the code (e.g. sparse matrix storage, matrix-vector multiplication,



and matrix inversion). As part of the computation, we need to compute the inverse of a weighted Laplacian matrix. The Laplacian matrix  $L = (\ell_{ij})$  of a undirected graph with weights  $w_e$  for all  $e \in E$  has  $\ell_{ij} = \ell_{ji} = -w_e$  for  $e = (i, j)$ , and  $\ell_{ii} = \sum_{j:(i,j) \in E} \ell_{ij}$ . The algorithm we use is as follows, where we set  $\epsilon = 0.01$ :

1. Set all  $\tilde{\gamma}_e$  values to 0, and let  $z_e = z_e^*$ .
2. Define  $q_e(\tilde{\gamma})$  to be the probability that an edge  $e = (i, j)$  will be included in a spanning tree  $T \in \mathcal{T}$  that is sampled with probability proportional to  $\exp(\sum_{e \in T} \tilde{\gamma}_e)$ . We iteratively improve  $\tilde{\gamma}$  until there is no  $\tilde{\gamma}_e$  value such that  $q_e(\tilde{\gamma}) > (1 + \epsilon)z_e$ , according the following procedure:
  - Estimate each  $q_e(\tilde{\gamma})$  for  $e = (i, j)$  using the current  $\tilde{\gamma}_e$  values by first creating the weighted Laplacian matrix  $L$  of the graph where each edge has weight  $e^{\tilde{\gamma}_e}$  and inverting it. Then, calculate  $q_e(\tilde{\gamma})$  as  $\vec{x}^T L^{-1} \vec{x}$ , where  $\vec{x}$  is the vector with the  $i$ th and  $j$ th entries as 1 and  $-1$  respectively, and zeroes elsewhere.
  - Loop over  $\tilde{\gamma}$ , subtracting  $\delta_e$  from each edge  $e$  such that  $q_e(\tilde{\gamma}) > (1 + \epsilon)z_e$ , where

$$\delta_e := \log \left( \frac{q_e(\tilde{\gamma}) \left(1 - \left(1 + \frac{\epsilon}{2}\right) z_e\right)}{(1 - q_e(\tilde{\gamma})) \left(1 + \frac{\epsilon}{2}\right) z_e} \right).$$

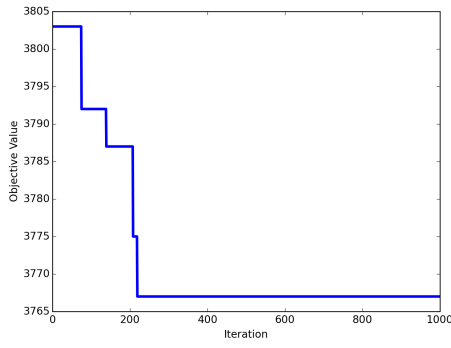
Once the  $\tilde{\gamma}_e$  have been computed, we need to be able to sample from the corresponding distribution. Asadpour et al. set  $\lambda_e = e^{\tilde{\gamma}_e}$ , and then sample a tree with probability proportional to  $\prod_{e \in T} \lambda_e$ . Asadpour et al. give an algorithm for computing such a sample in polynomial time, which we implemented. We also implemented the following algorithm for sampling  $\lambda$ -random trees used in the code of Oveis Gharan [24]: we pick an arbitrary starting node  $i$  as a location, and start with an empty edge set which will become the spanning tree. With probability proportional to the  $\lambda_e$  for each edge  $e = (i, j)$  incident on the current node  $i$ , pick an incident edge. If the node  $j$  has not yet been visited, add edge  $(i, j)$  to the tree. The new location becomes node  $j$ . Repeat until every node has been added to the tree (and hence the tree is spanning). This algorithm is not guaranteed to find a sample in polynomial time; however, we found that it scaled to larger instances better than the algorithm given in Asadpour et al.

We sampled trees as desired from the graph; as with the SwapRound method, 1000 samples were used for the results presented here. For each sampled tree, we ran Christofides' algorithm, and output the lowest cost result found across all samples. Once the  $\tilde{\gamma}$  are computed, we can draw the samples in parallel using four threads. The use of 1000 samples was not tuned, and we will consider another means for determining the number of samples to draw in the future. In Figure 2, we can see that in some cases drawing that many samples is not particularly useful in improving the outcome (although in one case it is).

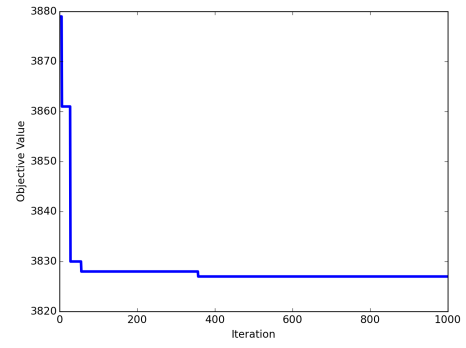
### 3 Experiments

The algorithms above were implemented in C++. We ran the algorithms on a machine with a 4.00Ghz Intel i7-875-K processor with 8GB DDR3 memory.

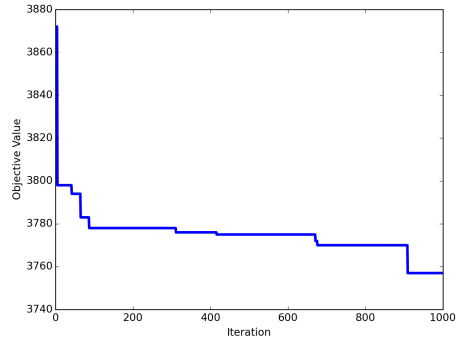
For test data, we used the TSPLIB instances of Reinelt [26]; we considered 59 two-dimensional Euclidean instances with up to 2103 vertices (averaging 524 vertices), and 5 non-Euclidean instances



(a) Maximum Entropy Sampling



(b) Column generation + SwapRound



(c) Splitting Off + SwapRound

Figure 2: Tradeoff between the number of samples drawn and the cost of the best tour found so far for the three different sampling methods on the VLSI instance XIT1083.

	Std	ColGen		ColGen+SR		MaxEnt		Split		Split+SR	
		Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave
TSPLIB (E)	9.56%	4.03%	6.44%	3.45%	6.24%	3.19%	6.12%	5.23%	6.27%	3.60%	6.02%
VLSI	9.73%	7.00%	8.51%	6.40%	8.33%	5.47%	7.61%	6.60%	7.64%	5.48%	7.52%
TSPLIB (N)	5.40%	2.73%	4.41%	2.22%	4.08%	2.12%	3.99%	2.92%	3.77%	1.99%	3.82%
Graph	12.43%	0.57%	1.37%	0.39%	1.29%	0.31%	1.23%	0.88%	1.77%	0.33%	1.20%

Table 1: Summary of results, giving the percentage in excess of optimal for the algorithms. ‘Std’ is Christofides algorithm, ‘ColGen’ is column generation, ‘MaxEnt’ is maximum entropy sampling, ‘Split’ is the splitting-off algorithm, and ‘SR’ is the swap round algorithm. The TSPLIB E instances are two-dimensional Euclidean, and the TSPLIB N instances are non-Euclidean.

with 120 to 1032 vertices (gr120, si175, si535, pa561, and si1032), averaging 484 vertices. Similarly, we considered 39 two-dimensional Euclidean VLSI instances of Rohe [27] with up to 3694 vertices, averaging 1473 vertices. We also considered instances of the graph TSP: we used 9 instances from the Koblenz Network Collection of Kunegis [16]. Specifically, we considered undirected simple graphs; if the graph had multiple connected components, we used the largest connected component and discarded the rest of the graph. The resulting instances ranged in size from 18 to 1615 vertices, averaging 363. Because these instances did not have optimal solutions previously computed for them, we used Concorde [4] to compute the optimal.

## 4 Results

A summary of our results can be found in Table 1. It shows the percentage above optimal for the algorithms mentioned averaged across the four different data sets we used, the Euclidean TSPLIB instances of Reinelt [26], the Euclidean VLSI instances of Rohe [27], the non-Euclidean TSPLIB instances, and the graph TSP instances from the Koblenz Network Collection [16], as described earlier. For the two methods that construct an explicit convex combination of spanning trees (column generation and splitting off), the best error is the error of the minimum-cost tour resulting from running Christofides’ algorithm over all the trees in the decomposition, while the average error is the average error from running Christofides’ algorithm over all the trees in the decomposition. For the methods that sample trees from a distribution (the swap round variants and the maximum entropy distribution), the best error is the smallest error found over all tours resulting from running Christofides’ algorithm on all the trees sampled from the distribution, and the average error is the average over all sampled trees. We make several observations based on this summary.

The first is that our results for Christofides’ algorithm are very similar to those found by Johnson and McGeoch [14], at least for the Euclidean TSPLIB and VLSI instances, with roughly 9-10% error; the error is less on the non-Euclidean TSPLIB instances. Somewhat surprisingly, Christofides’ algorithm seems to perform significantly worse on the graph TSP instances from the Koblenz Network Collection, with 12% error.

One reason that the performance of Christofides’ algorithm on the graph TSP instances is surprising is that for the other algorithms, the graph TSP instances seem to be significantly easier, with error under 1%. The VLSI instances appear to be the hardest overall for the algorithms collectively, but this may be because the average instance size is larger.

Another observation is that using SwapRound to sample trees does improve the overall performance of the output.

	Tree		Matching					
	Std	BOM	Std	ColGen	ColGen+SR	MaxEnt	Split	Split+SR
TSPLIB (E)	87.47%	98.57%	31.25%	11.43%	11.03%	10.75%	10.65%	10.41%
VLSI	89.85%	98.84%	29.98%	14.30%	14.11%	12.76%	12.78%	12.70%
TSPLIB (N)	92.97%	99.36%	24.15%	9.67%	9.36%	8.75%	8.77%	8.56%
Graph	79.10%	98.23%	39.31%	5.20%	4.84%	4.66%	4.34%	4.49%

Table 2: Costs of trees and matchings for the various methods, all expressed relative to the cost of the optimal tour.

Of all the algorithms, drawing from the maximum entropy distribution gives the best overall results, but constructing the convex combination via splitting off and then applying SwapRound was quite close in most cases, and better in some. Column generation is the worst of the variants, but we did not check whether the early termination of the column generation routine contributed to the weak performance of this variant.

Why are the Best-of-Many Christofides’ algorithm variants significantly better than Christofides’ algorithm? The key is that they trade off significantly higher spanning tree cost against significantly lower matching costs, with the reduction in the matching costs outweighing the increase in the spanning tree cost; these results are summarized in Table 2. The average tree cost for all of the Best-of-Many Christofides’ algorithm variants is at most the value of the subtour LP; the subtour LP is known to be very close to the cost of the optimal tour experimentally (about 98%-99% of optimal), and our experiments confirm this, while the minimum-cost spanning trees are 79%-93% of the cost of the optimal tour in our experiments. However, the matching costs are dramatically reduced. For Christofides’ algorithm, the cost of the matching is 25%-40% of the cost of the optimal tour, while for the Best-of-Many Christofides’ variants, it is 10%-15% in the case of the TSPLIB/VLSI instances, and 4-5% for the graph TSP instances.

One reason the matching costs are so much lower in the Best-of-Many Christofides’ algorithm variants is that sampling a spanning tree from the subtour LP solution gives spanning trees such that a very high percentage of the vertices have degree two. See Figure 3 to compare the minimum-cost tree on the VLSI instance XQF131 with trees produced by the Best-of-Many Christofides’ variants, and see Figure 4 to see a comparison of the degrees of the minimum-cost spanning tree versus a tree sampled from the maximum entropy distribution for the four different instance types (trees from other variants had degree distributions similar to the trees from the maximum entropy distribution). Thus the number of edges needed in the matching is much smaller. These edges tend to be longer in the Best-of-Many Christofides’ variants, because odd-degree vertices are rarer, and thus not as near to each other, but because the number of edges needed is much smaller, there is a significant reduction in the cost of the matching. We summarize information about the matching costs in Table 3, where we give the average fraction of odd-degree nodes for the various algorithms, as well as the average cost of a matching edge (expressed in terms of percentage of the cost of an optimal tour). For the TSPLIB and VLSI instances, 36 to 39% of the min-cost spanning tree vertices are odd, whereas for the Best-of-Many variants, the number is between 8-12%; however, the cost of each matching edge is roughly half to two-thirds more in these instances. The graph instances are quite different; for these instances, the min-cost spanning trees have nearly 66% of vertices having odd degree, while the Best-of-Many variants have about the same percentages as before. For these instances, however, the cost of the matching edges are about the same for the standard Christofides’ algorithm and the Best-of-Many variants.

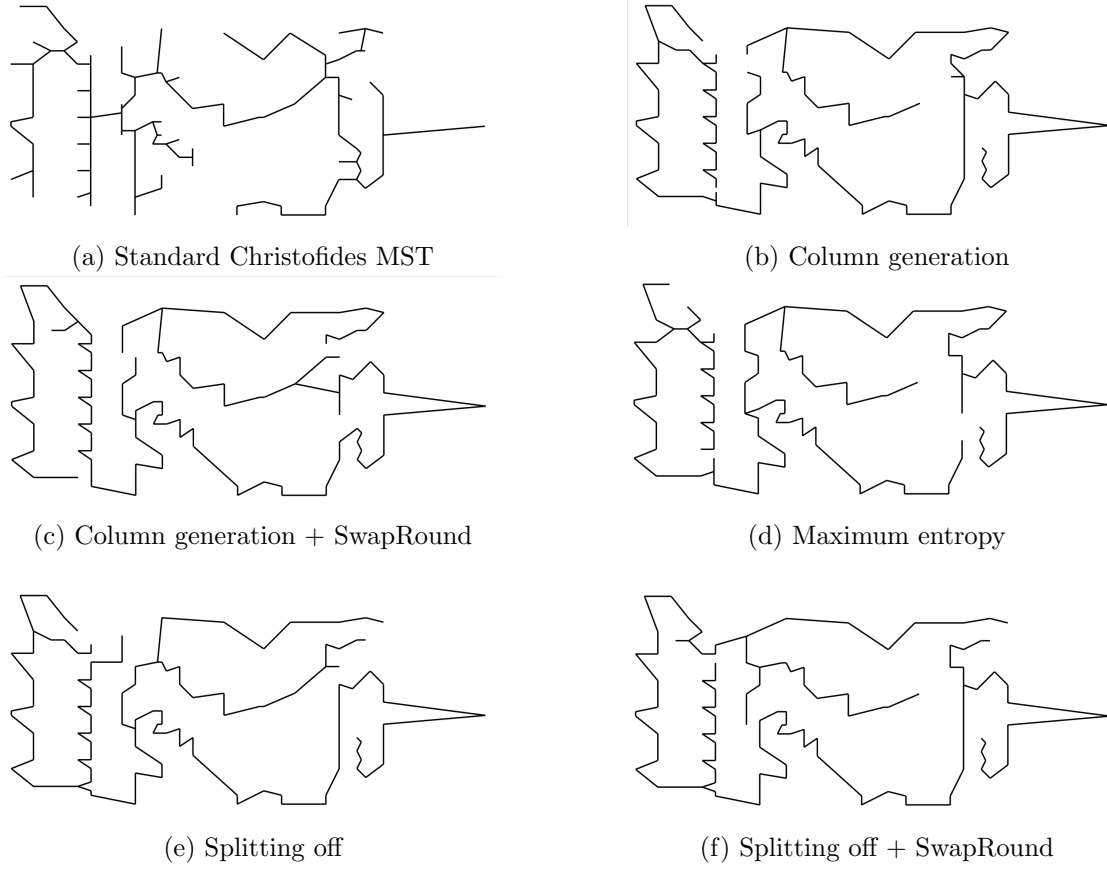
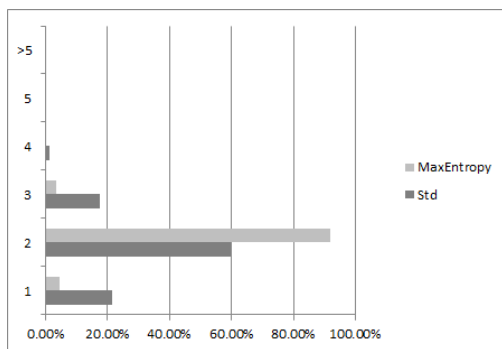


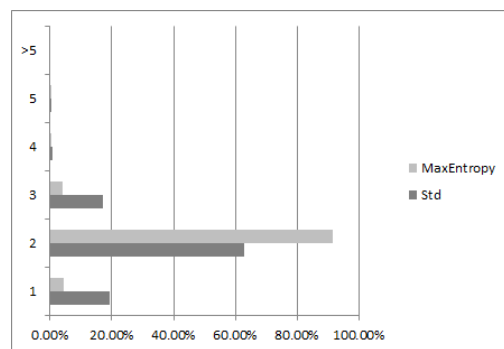
Figure 3: Sample trees from the various methods on VLSI instance XQF131 from Rohe [27].

	Std		ColGen		ColGen+SR		MaxEnt		Split		Split+SR	
	Num	Cost	Num	Cost	Num	Cost	Num	Cost	Num	Cost	Num	Cost
TSPLIB (E)	39%	0.89%	9.3%	1.3%	8.4%	1.4%	8.0%	1.4%	7.6%	1.5%	7.8%	1.4%
VLSI	36%	0.21%	12%	0.34%	11%	0.36%	8.6%	0.38%	8.3%	0.41%	8.3%	0.39%
TSPLIB (N)	38%	0.48%	12%	0.63%	11%	0.67%	10%	0.66%	9.8%	0.69%	9.8%	0.66%
Graph	66%	1.9%	8.9%	1.7%	8.2%	1.7%	7.8%	1.7%	7.6%	1.5%	7.8%	1.5%

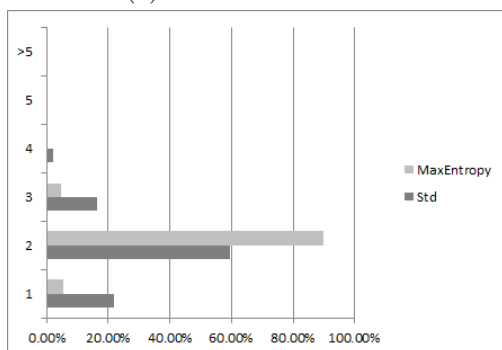
Table 3: Information about matchings for the various algorithms and instances. ‘Num’ indicates the average percentage of odd-degree vertices in the spanning tree, and ‘Cost’ is the average cost of a matching edge expressed as a percentage of the cost of the optimal tour.



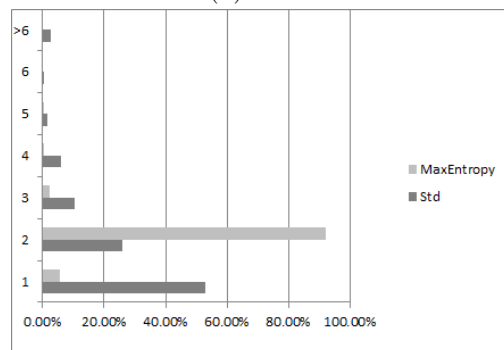
(a) Euclidean TSPLIB



(b) VLSI



(c) TSPLIB Non-Euclidean



(d) Graph

Figure 4: Degree distributions of the minimum-cost spanning tree (used in standard Christofides) versus a tree sampled from the maximum entropy distribution.

	Std	ColGen	ColGen+SR	MaxEnt	Split	Split+SR
TSPLIB (E)	0.1	82.2	6900.1	1335.2	3470.0	6671.7
VLSI	0.2	208.7	10414.4	27532.3	18128.2	29578.7
TSPLIB (N)	0.1	177.3	376.4	199.1	25.5	48.5
Graph	1.4	21.0	3401.1	292.1	2556.0	2821.6

Table 4: Average running times (user time) for the variants on the different instance types. The running times do not take into account the time to compute the Subtour LP solution, although this was usually quite fast.

	1	2	3	4	5	6	> 6
Random	0.221	0.566	0.206	0.007	0.000	0.000	0.000
TSPLIB (E)	0.214	0.597	0.175	0.014	0.000	0.000	0.000
VLSI	0.192	0.627	0.171	0.170	0.010	0.000	0.000

Table 5: A comparison of the degree distribution from MSTs on random Euclidean instances (drawn from simulations performed by Steele, Shepp, and Eddy [33, Table 1]) and the degree distribution of MSTs computed on the Euclidean instances from TSPLIB and the VLSI instances.

Although we did not try to implement the various algorithms to be as fast as possible, it is instructive to compare their running times: see Table 4. Note that the reported running times do not include the time taken to solve the Subtour LP, although this was usually quite fast. The standard Christofides’ algorithm is quite fast compared to the other algorithms that must run Christofides’ algorithm many times in addition to doing significant extra computational work.

As a final observation, Schalekamp [28] has pointed out that our degree distributions for the MSTs computed for Christofides’ algorithm on Euclidean instances looks similar to the degree distributions for MSTs from random Euclidean instances. In particular, suppose we draw points  $n$  uniformly at random from the unit square, and compute the MST on these points. Steele, Shepp, and Eddy [33, Theorem 2] have shown that as  $n$  tends to infinity, the fraction of nodes of degree  $k$  in the tree has a constant limit with probability 1. In Table 5, we compare the degree distribution from these random instances that Steele et al. obtained via simulation with the average degree of MSTs computed on the two sets of Euclidean instances. For graph TSP instances, we can also compare degree distributions with the following model: suppose we compute an MST in a complete graph in which edge costs are drawn uniformly from  $[0,1]$ . Aldous [1, Proposition 2] gives a closed formula for the degree of a specified vertex in such a tree as  $n$  tends to infinity. In Table 6, we compare the degree distribution from MSTs in this random model with the average degree of the MST computed in the graph TSP instances. These appear to be less similar to the random instances; the fraction of degree one vertices is lower in the random instances, while the fraction of degree two and three vertices is higher. Given the small number of graph TSP instances we use, it is hard to know if the results would be closer if we ran a larger number of instances, or whether there is some fundamental reason the two distributions are different.

	1	2	3	4	5	6	> 6
Random	0.408	0.324	0.171	0.068	0.022	0.006	0.001
Graph	0.529	0.260	0.104	0.059	0.015	0.006	0.026

Table 6: A comparison of the degree distribution from MSTs on complete graphs with random weights (drawn from a computation of Aldous [1, p. 396]) and the degree distribution of MSTs computed on our graph TSP instances.

## 5 Conclusions

Our goal in this paper was to determine whether the empirical performance of the Best-of-Many Christofides’ algorithms gives any reason to think they might be provably better than the Christofides’ algorithm. The answer to this question appears to be yes, with the large caveat that there are many heuristics for the traveling salesman problem (like Lin-Kernighan) with far better performance than Christofides’ algorithm which have no provable performance guarantee at all. We also wished to determine which variant might be most promising for further theoretical study. For this question, it seems that the sampling methods have the most promise; that is, maximum entropy sampling or the SwapRound algorithm applied to some initial convex combination of trees. However, because the good performance of these algorithms depends on taking the best result over a large number of samples drawn, one might have to argue that a good tour is produced with reasonable probability after multiple draws; it does not seem that one can argue that a good tour is produced in expectation from a single draw. Because the average performance of these sampling methods does not seem significantly different from what happens when we construct an explicit convex combination (with column generation, or splitting off), it might be about as easy to prove that the expected value of a single draw of a sampling method is good as it is to prove that the average tour generated from an explicitly given convex combination is good, for a carefully constructed combination.

Very recently, Schalekamp and van Zuylen [29] created an example that shows that for an *arbitrary* decomposition of an LP solution into a convex combination of 1-trees, it is not possible for the Best-of-Many Christofides’ algorithm to do better than a  $\frac{3}{2}$ -approximation algorithm; we show the example in Figure 5. A *1-tree* is a set of edges that has two edges incident to an arbitrary vertex, along with a spanning tree on the remaining vertices. Held and Karp [12] show that the Subtour LP solution can be decomposed into a convex combination of 1-trees. This example provides further evidence that the randomization inherent in either maximum entropy sampling or SwapRound may well be needed to overcome worst-case decompositions. Vygen’s [34] recent paper for the *s-t* path TSP also uses the idea of modifying an initial decomposition of spanning trees so that the decomposition has specific properties that allow for a better performance guarantee than otherwise could be obtained.

In our limited experience, it seems that graph TSP is a significantly easier problem for the Best-of-Many variants, and this may bear further investigation both theoretically and empirically. The quality of the solutions found by these algorithms relative to the standard Christofides’ algorithm may indicate that these instances really are much easier than the general case of symmetric cost functions with triangle inequality.



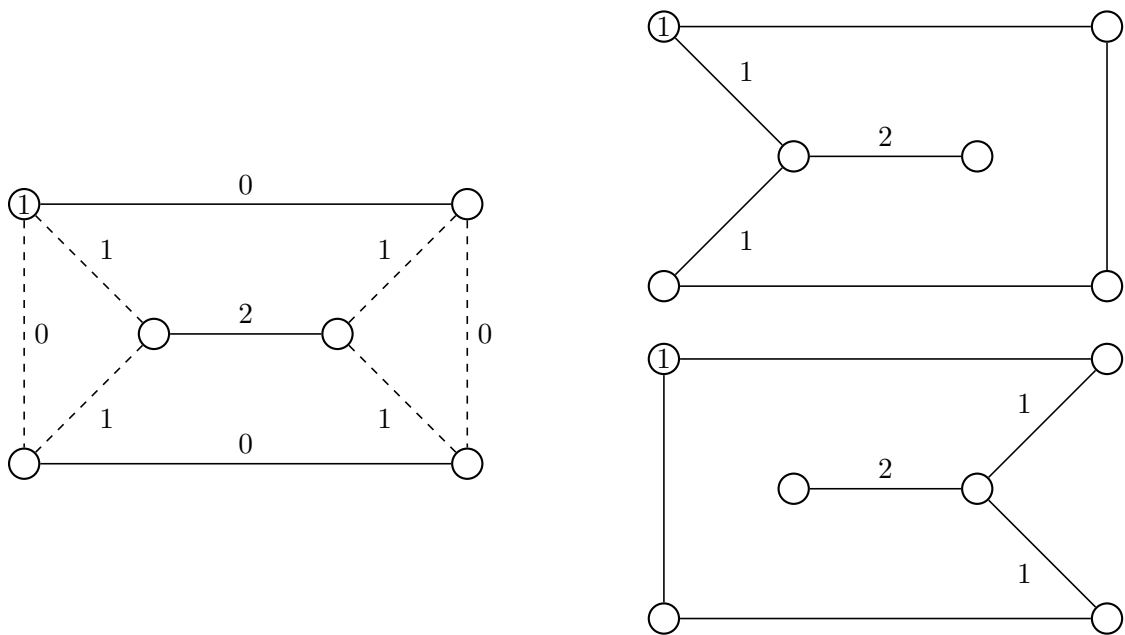


Figure 5: Example of Schalekamp and van Zuylen; the LP solution is on the left, and the convex combination of 1-trees is on the right, with each tree having weight  $1/2$ . In the LP solution, dashed lines indicate LP value of  $1/2$ , solid lines indicate LP value of  $1$ . Numbers represent the cost of the edges; other edge costs are implied by the triangle inequality. The LP solution has a cost of  $4$ , and any matching of the odd-degree vertices in the decomposition will have cost at least  $2$ , so that the result of Christofides' algorithm will cost at least  $3/2$  times the cost of the LP solution. One can verify that the LP solution is optimal via a duality argument.

## Acknowledgments

We thank Shayan Oveis Gharan for generously sharing his maximum entropy code with us; we used many of his implementation ideas in coding our own algorithm. We thank Frans Schalekamp and Anke van Zuylen for allowing us to include their example of a bad case for Best-of-Many Christofides. We thank Frans Schalekamp for several observations which he allowed us to use and include.

## References

- [1] D. Aldous. A random tree model associated with random graphs. *Random Structures and Algorithms*, 1:383–402, 1990.
- [2] H.-C. An. *Approximation Algorithms for Traveling Salesman Problems Based on Linear Programming Relaxations*. PhD thesis, Department of Computer Science, Cornell University, Aug. 2012.
- [3] H.-C. An, R. Kleinberg, and D. B. Shmoys. Improving Christofides’ algorithm for the  $s$ - $t$  path TSP. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 875–886, 2012.
- [4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde 03.12.19. Available at <http://www.math.uwaterloo.ca/tsp/concorde/index.html>.
- [5] A. Asadpour, M. X. Goemans, A. Madry, S. Oveis Gharan, and A. Saberi. An  $O(\log n / \log \log n)$ -approximation algorithm for the asymmetric traveling salesman problem. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 379–389, 2010.
- [6] C. Chekuri, J. Vondrák, and R. Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 575–584, 2010. See full version at [arxiv.0909:4348](https://arxiv.org/abs/0909.4348).
- [7] N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.
- [8] Eigen. Available at <http://eigen.tuxfamily.org/index.php>. Accessed January 28, 2015.
- [9] A. Frank. *Connections in Combinatorial Optimization*. Oxford University Press, Oxford, UK, 2011.
- [10] A. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12:23–39, 1982.
- [11] Z. Gao. On the metric  $s$ - $t$  path traveling salesman problem. CORR abs/1404.7569, 2014.
- [12] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1971.

- [13] J. A. Hoogeveen. Analysis of Christofides’ heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10:291–295, 1991.
- [14] D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variants*, pages 369–443. Kluwer Academic Publishers, 2002.
- [15] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009. Code available at <http://pub.ist.ac.at/~vnk/software.html>.
- [16] J. Kunegis. KONECT – the Koblenz network collection. In *Proceedings of the International Web Observatory Workshop*, pages 1343–1350, 2013.
- [17] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973.
- [18] L. Lovász. On some connectivity properties of Eulerian graphs. *Acta Math. Acad. Sci. Hungar.*, 28:129–138, 1976.
- [19] T. Mömke and O. Svensson. Approximating graphic TSP by matchings. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, pages 560–569, 2011.
- [20] M. Mucha. 13/9-approximation for graphic TSP. *Theory of Computing Systems*, 55:640–657, 2014.
- [21] H. Nagamochi and T. Ibaraki. Deterministic  $\tilde{O}(mn)$  time edge-splitting in undirected graphs. *Journal of Combinatorial Optimization*, 1:5–46, 1997.
- [22] G. Optimization. Gurobi 5.6.3. Available at [www.gurobi.com](http://www.gurobi.com), 2014.
- [23] S. Oveis Gharan. *New Rounding Techniques for the Design and Analysis of Approximation Algorithms*. PhD thesis, Department of Management Science and Engineering, Stanford University, May 2013.
- [24] S. Oveis Gharan. Personal communication, 2014.
- [25] S. Oveis Gharan, A. Saberi, and M. Singh. A randomized rounding approach to the traveling salesman problem. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, pages 550–559, 2011.
- [26] G. Reinelt. TSPLIB – a traveling salesman problem library. *ORSA Journal on Computing*, pages 376–384, 1991.
- [27] A. Rohe. Instances found at <http://www.math.uwaterloo.ca/tsp/vlsi/index.html>. Accessed December 16, 2014.
- [28] F. Schalekamp. Personal communication. 2015.
- [29] F. Schalekamp and A. van Zuylen. Personal communication, 2015.

- [30] A. Sebő. Eight-fifth approximation for the path TSP. In M. X. Goemans and J. R. Correa, editors, *Integer Programming and Combinatorial Optimization - 16th International Conference, IPCO 2013*, volume 7801 of *Lecture Notes in Computer Science*, pages 362–374. Springer, 2013.
- [31] A. Sebő and J. Vygen. Shorter tours by nicer ears:  $7/5$ -approximation for the graph-TSP,  $3/2$  for the path version, and  $4/3$  for two-edge-connected subgraphs. *Combinatorica*, pages 1–34, 2014.
- [32] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 1996.
- [33] J. M. Steele, L. A. Shepp, and W. F. Eddy. On the number of leaves of a Euclidean minimal spanning tree. *Journal of Applied Probability*, 24:809–826, 1987.
- [34] J. Vygen. Reassembling trees for the traveling salesman. *CoRR*, abs/1502.03715, 2015.